Automatic Synthesis of the ZRTP Protocol Guide

Student : Di Long, Li (Darren)
Supervisor : Rajeev Gore
Course code : COMP3740

Australian National University
2017 Oct 23rd
Introduction

- Encode ZRTP in CryptoVerif
- Draws comparison with ProVerif encoding of ZRTP done by others
Project scope

- Tamarin was originally included as part of the overview
- Dropped due to language complexity and setbacks in CryptoVerif encoding
- Scope limited to ProVerif and CryptoVerif encoding of ZRTP
Project at a glance

- Take ZRTP apart (implement individual mechanisms of ZRTP in CryptoVerif separately)

- Fixes

- Result
  - It runs
  - Proof of authentication does not rely on core properties of ZRTP
  - Still fails
Preliminaries

- Key exchange algorithms
- Key exchange protocols
- ZRTP
- Proverif and CryptoVerif
Key exchange algorithms

- Key exchange algorithms
  - Diffie-Hellman
- Allows two parties to exchange partial public information to reach a state with shared secrets
- Basis of key exchange protocols
- MiTM attacks still possible due to lack of authentication – normally not used on its own
Key exchange protocols

- Based on key exchange algorithms
- Adds authentication
- Normally require preexisting cryptographic materials for signatures of some form
  - Certificates
  - Public & private keys
ZRTP

- Key exchange without existing cryptographic materials
- Uses Short Authentication String as an important part of the verification mechanism[1]
- Hash commitment mechanism allows SAS to be viable[1]

Role of ZRTP

- ZRTP session to reach to state with shared secrets
- Obtained shared secrets are used to encrypt SRTP calls
- SRTP is encrypted voice, video call over IP
- Overall:
  - ZRTP(key exchange) → SRTP(voice, video call over IP)
ProVerif & CryptoVerif
Input files

- ProVerif takes pi-calculus, among other input formats[2]
  - Pi-calculus allows reasoning of concurrent processes
- CryptoVerif takes process calculus as input[3]
  - Process calculus is based on pi-calculus
  - But is always typed

ProVerif & CryptoVerif
Query system

- Both uses a query system to “ask” prover to prove some properties or predicate
- Both can prove
  - Secrecy
  - Authentication (via event correspondence)
ProVerif & CryptoVerif

Adversary model

- Full control over network
  - Can intercept, modify any messages
  - May send intact messages to intended destination
  - May send corrupted messages to intended destination
  - May drop messages
ProVerif & CryptoVerif

Adversary model

- Capable of computation
- Tries to gather information – break secrecy
Cryptographic primitives Modelling

- Symbolic model (ProVerif)
  - Cryptographic operations are defined by functions/constructors and destructors [2][4]
  - Cannot model cryptographic primitives of different strength

- Computational model (CryptoVerif)
  - Allows reasoning with computational assumptions, probability of breakage, etc[3]

- Computational model is more realistic

Defining hash and encryption (ProVerif)

fun hash(bitstring) : bitstring.

fun encrypt(bitstring, bitstring) : bitstring.

reduc
    forall x : bitstring, y : bitstring;
    decrypt(encrypt(x,y),y) = x.
Defining hash and encryption (CryptoVerif)

type block [fixed,large].
...(other type definitions)

expand ROM_hash(hashkey, block, block, block, hash).
expand IND_CPA_sym_enc(keyseed, block, block, block, encseed, keygen, encrypt, decrypt, injbot, zeros, Penc).

fun hash_symbolic (hashkey, block) : block [compos].
Notion of secrecy

- Symbolic model
  - Weak secrecy
    - Attacker cannot derive exactly the information of interest[4]
  - Strong secrecy
    - Attacker has zero knowledge of the information (not even partial)[4]
- Computational model
  - Attacker has negligible probability of distinguishing the information from a random value[4]
Adversary Cryptographic capability

- Symbolic model (ProVerif)
  - Limited to defined cryptographic primitives\textsuperscript{[4]}
- Computational model (CryptoVerif)
  - Adversary is a probabilistic polynomial-time Turing machine\textsuperscript{[4]}
Project in detail

Study of ZRTP specs

- Core security components are identified
  - Diffie-Hellman
  - Hash commitment
  - Short Authentication String (SAS)
    - 16 bit (4 hex characters) string
- Interesting aspects
  - The above mechanisms combined gives secrecy and authentication in first run
Project in detail
Translation work

- A review and cleanup was done on ProVerif encoding of ZRTP (by R. Bresciani and A. Butterfield[5])
- Translated from untyped pi-calculus to typed pi-calculus
- Formatting

Project in detail

Encode ZRTP in CryptoVerif

- Use ProVerif code and ZRTP specification as basis
  - Picked up minor mistakes of the ProVerif code
- Slight difference in directions
  - ProVerif code reasons for second run (and onwards) of ZRTP
  - CryptoVerif code (tries) to reason for first run of ZRTP
Project in detail
Encode ZRTP in CryptoVerif

- Verbal verification of SAS is modelled by
  - Signed and encrypted exchange of SAS
  - Check for mismatch
Project in detail
Encode ZRTP in CryptoVerif

• Results
  – Runs very slowly
  – Fails with stack overflow of the CryptoVerif program
Project in detail

Take ZRTP apart

- Implement individual mechanisms of ZRTP in CryptoVerif separately
- HMAC
- Symmetric encryption
- Hash images
  - Causes CryptoVerif to run very slowly
  - Not sure if terminates
Project in detail

Remedy

- Hash in Random Oracle Model was used for first version
  - Chained usage of it causes major slowdown
- Replaced by symbolic definition (similar to one in ProVerif)
Results

- Runs – finishes in seconds
- Can prove authentication (not useful – more on next slide)
- Cannot prove secrecy of shared secret
Results

CryptoVerif output

RESULT Proved event testI1 ==> testR1 up to probability
Psigntime(context for game 7) + time, 1.)

RESULT time(context for game 7) = time(keygen) + (nH + 13.) *
time(hash) + time(serialiseConfirm2) + time(skgen) + time(sign) + 6. *
time(hash_symbolic) + 4. * time(serialiseHello) + 2. *
time(serialiseDHPart1) + 2. * time(serialiseCommit) + 4. *
time(checkhmac) + 2. * time(concat) + 4. * time(exp) + 4. *
time(serialiseHellowMAC) + 2. * time(serialiseCommitwMAC) + 2. *
time(serialiseDHPart1wMAC) + 4. * time(serialiseDHPart2wMAC) + 2. *
time(concatForTotalHash) + 2. * time(concatFors0) + 2. *
time(serialiseConfirm1) + 3. * time(let injbot) + 3. * time(decrypt) + 2. *
time(serialiseDHPart2) + 13. * time(hmac) + 4. * time(encrypt)

... (ignored)
Results
Shortcomings

- Proof of authentication relies on the modelling of SAS verbal verification
- Probability of attack is reduced to probability of breaking signature
  - Not intended!
- Ignores probability of attack in relation to SAS length (the original main goal)
Conclusion

• Modelling first run can be difficult
  – Incorrect modelling of SAS verification leads to unintended proof results
• Modelling of secrecy in computational model was not intuitive
Limitations

● Models are still models
  – Neither of the provers take side channel attacks into account
● e.g. Stealing keys from computers, mobiles via physical access[6][7][8]
● Normal “gotchas” of proofs apply
  – System assumed to be OK may not be OK

Future directions

- Reason with SAS verification in external theorem provers (e.g. Coq)
- Use frameworks based on theorem provers (e.g. EasyCrypt)
- Only formally analyse second run (and onward) in CryptoVerif